Process-based assessment of computer science students

Walter Alvarez Grijalba
Escuela de Ingeniería en Computación
Centro Académico de Alajuela
Tecnológico de Costa Rica
Alajuela, Costa Rica
walvarez@ic-itcr.ac.cr

José Arguedas Castillo
Escuela de Ingeniería en Computación
Centro Académico de Alajuela
Tecnológico de Costa Rica
Alajuela, Costa Rica
joarguedas@ic-itcr.ac.cr

Randall Araya Campos
Escuela de Ingeniería en Computación
Centro Académico de Alajuela
Tecnológico de Costa Rica
Alajuela, Costa Rica
raraya@ic-itcr.ac.cr

Paula Estrella
Facultad de Lenguas
Universidad Nacional de Córdoba
Córdoba, Argentina
pestrella@gmail.com

***Abstract*—Keylogging tools are frequently used to monitor and evaluate writing processes. In this work we propose to use a keylogger to compare advanced and novice computer science students' programming process during experimental sessions. The main goal is to validate if it is possible to estimate how much effort a student requires to solve a specific programming problem. The most significant contributions of this study are: the improvement of an open source keylogging, the implementation of an analysis tool to process and show the collected data in a non-technical way, and the proposal of a new set of indicators for measuring a student's programming effort. The results obtained in the experiments carried out are promising and in many cases confirm the hypotheses.**

***Index Terms*—Programming, Process-based, Monitoring, Keylogger, Effort metric, Effort indicator.**

I. Introducción

En la actualidad cuando se evalúan las capacidades de un estudiante de computación para crear programas, es común tomar en cuenta solamente las funciones que realizó, qué tan legible es el código del programa y el resultado final de ejecutarlo [1], [2]. Los estudiantes son evaluados por el desempeño de su *producto*, si este no realiza lo que se solicitó, la nota que ellos reciban será penalizada o no obtendrán nota alguna, sin importar cuanto esfuerzo le ha significado al alumno. Sin embargo, el desarrollo de software es una suma de *procesos* cognitivos que comprende más que lo que vemos en el producto final: para un estudiante significa horas de investigación y no queda explícito si se copió código (y cuánto), cuántas líneas se eliminaron o cuánto se investigó para generar el producto final. Además, escribir el código de un programa no es un proceso continuo en el cual el programador se sienta en frente de su computadora y genera de manera espontánea un programa, sino que durante este proceso surgen dudas e inconvenientes, que el programador deberá resolver ya sea realizando búsquedas, buscando material de apoyo o aludiendo a su capacidad intuitiva.

¿Qué pasaría si pudiéramos monitorear a los programadores durante una tarea de programación? Probablemente seríamos capaces de saber el grado de dificultad que representa un problema dado para un programador, cómo éste analiza e implementa una solución y cuánto tiempo representa en su línea de producción el resolver ese problema, lo cual puede considerarse el *esfuerzo de programar*. La forma más sencilla sería observando al sujeto mientras realiza una tarea de programación y tomando apuntes de cómo se desempeña (qué búsquedas hace, cuántas veces compila, qué tipo de errores comente, etc), sin embargo esta estrategia rápidamente se vuelve inviable cuando la cantidad de sujetos aumenta, como es el caso de una clase estándar de programación. Una mejor estrategia, con mayor validez ecológica (es decir, que introduce menos subjetividades como las anotaciones de un investigador y es menos invasiva para el sujeto) es el uso de herramientas de recolección de datos de teclado, de mouse y de actividad en la pantalla (denominados keylogging y screenrecording, respectivamente), lo cual es completamente transparente al usuario.

Anteriormente, se ha explorado el proceso y el producto utilizando keylogging con alumnos de grado mientras generan traducciones en lo que conoce como *estudio del proceso traductor* [3], [4]. El proceso traductor, definido en la sección II, tiene similitudes con el proceso de programación en tanto en ambos casos hay comprensión y generación de lenguaje (ya sea formal o natural). Bajo esta hipótesis es que el objetivo de este trabajo es migrar y aplicar la metodología del proceso traductor al *proceso programador* en alumnos de ciencias de la computación tanto para explorar la factibilidad de utilizar la misma herramienta de keylogging y sus aplicaciones de análisis de datos, como para intentar proponer una forma de evaluar a los alumnos que considere no sólo el producto sino también el proceso, en lo que denominamos *esfuerzo de desarrollar un programa*.

II. Procesos de escritura: Traducción y Programación

En trabajos previos (no citados en este artículo) hemos estudiado una tarea de producción de lenguaje natural específica, que es la traducción, y en el presente trabajo intentaremos replicar la metodología para otro proceso de escritura como es la programación. En las siguientes subsecciones explicaremos las similitudes entre estos procesos.

II-A. Proceso de traducción

Se puede entender la traducción como la actividad que consiste en comprender el significado de un texto en un idioma para producir un texto con significado equivalente, en otro idioma. Según [5], en este proceso pueden identificarse tres fases que se ilustran en la Figura 1: *Orientación:* desde que el traductor empieza a leer el texto fuente hasta que presiona la primer tecla para empezar a formar el texto traducido. *Elaboración de un borrador:* Comienza inmediatamente luego de la etapa anterior y finaliza cuando el traductor presiona el punto final de la primera versión completa del texto traducido. *Revisión:* Inicia inmediatamente luego de la etapa anterior y finaliza cuando el traductor determina que la traducción es satisfactoria y final. Tiene como objetivo determinar si la traducción tiene la calidad deseada por el traductor y realizar los cambios correspondientes hasta que la traducción pase los criterios de calidad del traductor.

II-B. Programación como una tarea de producción escrita

Si bien las definiciones provistas al comienzo de esta sección se aplican al lenguaje natural, hay evidencia de que se pueden extrapolar a la programación. Desde la aparición de lenguajes de programación de más alto nivel, han surgido áreas de investigación que relacionan la psicología, la cognición y las computadoras [6]. Un trabajo fundacional por Curtis et al. (1986) explica que "*[...] Un programa es también un texto. Los programas pueden ser entendidos y ejecutados por un computadora, y pueden ser entendidos y modificados por otros programadores. Así, es fácil de afirmar que existe un paralelismo entre la comprensión del programa y comprensión del texto del lenguaje natural. Dado que el texto ha existido por más tiempo, la investigación sobre sus procesos está más desarrollada que la investigación en la programación. [...] Las métricas de software utilizadas en la programación son equivalentes a las fórmulas de legibilidad utilizadas en la investigación sobre texto.*" [7].

Pensando en el ciclo de vida del desarrollo de software y haciendo una analogía con el proceso traductor, proponemos que las fases corresponderían al análisis y diseño, implementación y testeo, respectivamente, como se ilustra en la Figura 1.

Esto nos permite plantearnos el estudio de las competencias de los alumnos de computación en términos similares a los que se estudian otros procesos de escritura. Concretamente, nos interesa incluir indicadores del proceso a las formas más clásicas de evaluar estas competencias. Hay algunos trabajos que proponen esto para propiciar el aprendizaje basado en práctica, para proveer retroalimentación instantáneo a medida que se observan las acciones del programador, como por ejemplo en [8]–[10].

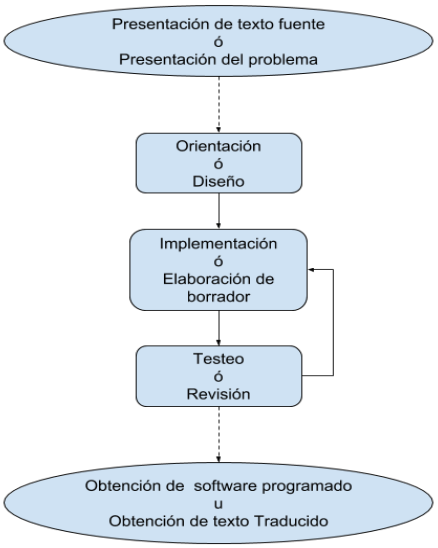

Figura 1. Fases en la producción de software.

La problemática de la enseñanza de programación es un tema central en la conferencia LACLO y existe una vasta literatura al respecto, que por motivos de espacio no cubriremos en más detalles en este artículo.

III. Monitoreo de la generación de código

Un método popular para supervisar el proceso de escritura es mediante el uso de un *keylogger*, un programa que registra toda la actividad del teclado y el mouse llevada a cabo por el usuario de una computadora, y también puede registrar marcas de tiempo para cada tecla y para períodos inactivos (pausas) durante una sesión de escritura. Los keyloggers se han utilizado desde los años 90 y desde entonces, se han desarrollado varias herramientas. Sin embargo, para el caso puntual del estudio del proceso traductor (nuestra investigación original), las herramientas disponibles no funcionaban bajo Linux, entorno disponible en los laboratorios de la UNC. Eso motivó el desarrollo de un nuevo keylogger, denominado *ResearchLogger* [11], que se está implementando actualmente como un paquete de código abierto y que se probó inicialmente en un estudio piloto con alumnos de traducción para investigar los procesos cognitivos involucrados en la traducción humana y actualmente se está probando con alumnos principiantes y avanzados de computación para estudiar la evaluación de tareas de programación basadas en el proceso como como complemento a lo que generalmente se evalúa que es el producto.

El *ResearchLogger* es la nueva herramienta que se generó a partir de adaptar y extender otra herramienta de código abierto existente, y acompañando este paquete de toma de datos se desarrolló uno de análisis de los datos recolectados, denominado *ResearchAnalyzer*. Con el *ResearchLogger* podemos

observar las acciones de un usuario en una sesión, es decir, permite la observación de las aplicaciones que utiliza, los recursos online que consulta, las búsquedas que realiza, etc y cada evento de teclado o mouse se registra con precisión de milisegundo en un archivo de salida que denominamos *log*. Por lo tanto, el *log* detallado nos permite saber cómo el sujeto distribuyó el tiempo para cada tarea, cuando hubo pausas de escritura, entre otros.

Inicialmente para cada sesión de trabajo del *ResearchLogger* genera 4 carpetas de salida *click_images*, *detailed_log*, *system_log*, y *timed_screenshots*. En la primera carpeta se encuentran las imágenes tomadas durante la sesión junto con un archivo de texto descriptivo de las imágenes, en la segunda carpeta se encuentra un archivo de texto descriptivo de las interrupciones de teclado y mouse captadas durante la sesión llamado *'detailedlog_<nombre de usuario>'*, que será empleado para realizar el análisis total; y finalmente, en la carpeta restante se encuentra un archivo de texto con relación a la ejecución interna del programa *Researchlogger*, con descripciones de eventualidades en tiempo de ejecución.

El primer aporte de este trabajo ha sido agregarle una funcionalidad básica para que sea más accesible a usuarios no técnicos. El objetivo es facilitar su instalación, para lo cual realizamos un instalador de la herramienta para el sistema operativo Microsoft Windows, ya que inicialmente se tenía que clonar un repositorio de GitHub y posteriormente instalar las dependencias de APIs manualmente; actualmente en cambio, se cuenta con un instalador en 2 pasos para las siguientes distribuciones Windows XP, Windows 7, Windows 8 y Windows 10. En siguientes etapas de desarrollo se prevé un instalador para otros sistemas operativos.

La herramienta *ResearchAnalyzer* fue diseñada para realizar un análisis preliminar de la información obtenida en los *logs*, realizando un análisis estadístico de las interrupciones de mouse y teclado que se imprime tanto en consola como texto plano. El segundo aporte de este trabajo consiste en haber agregado dos funcionalidades más a al *ResearchAnalyzer*: una para verificar la compatibilidad de *logs* creados en distintas plataformas y la más importante para crear un reporte con el análisis realizado para que el usuario del sistema no deba lidiar con cuestiones técnicas no relacionadas directamente al análisis de *logs*. Estas funcionalidades fueron modeladas en python, como se describe a continuación.

El primer módulo *'scriptchecker.py'* realiza un chequeo preliminar de los *logs* generados en Windows, ya que inicialmente el *ResearchAnalyzer* fue diseñado y testeado con *logs* generados en Ubuntu (una distribución del sistema operativo Linux) y al recolectar datos en Windows detectamos algunas diferencias propias de la información de bajo nivel que no se puede obtener en Windows; el chequeo revisa la estructura del *log* generado y lo transforma para que no genere inconvenientes a la hora de cargar los datos al software de análisis.

El segundo módulo *'logsinterpreter.py'* realiza un análisis más específico del *log* obtenido de la carpeta *detailed_log* y permite visualizar los datos en un reporte generado en un formato compatible con hojas de cálculo (MS Excel, LibreOffice Calc, entre otros). Este módulo permite analizar una sesión o analizar y comparar varias sesiones a la vez.

Dicho análisis esta compuesto por las siguientes partes:

- Distribución de tiempo en cada herramienta utilizada por un estudiante.
- Cantidad de veces que un estudiante realiza una compilación.
- Tiempo total de la sesión de programación.
- Búsquedas Realizadas en web, este filtrado se realiza con la ayuda del tag *'Buscar con Google - Google Chrome'*.
- Páginas consultadas en web, este filtrado se realiza con la ayuda del tag *'- Google Chrome'*.
- Una reconstrucción parcial de texto realizado por los estudiantes a partir del *log* de las interrupciones de teclado.
- Tiempo por recurso, esto se presenta en la reconstrucción parcial de texto.

Cuando se genera una visualización de los datos se le asigna nombre de la siguiente manera: *'Analisys_<Código de estudiante>_<Código de sesión de programación>'* si solamente se esta analizando una sesión de programación y en caso de analizar varias sesiones se genera para cada sesión un archivo como el anterior y adicionalmente se genera otro archivo llamado *'Sessions_Analysis'* que contiene comparaciones de datos entre las diferentes sesiones; todos los archivos finalmente son ubicados en una carpeta llamada *'Analisys'*.

IV. Estudio propuesto

Luego de implementar las mejoras mencionadas en la Sección III, proponemos un estudio experimental con los siguientes objetivos:

- Evaluar las nuevas funcionalidades del *ResearchLogger* y *ResearchAnalyzer* como herramientas apropiadas para estudiar el proceso de programación.
- Estudiar el proceso de programación mediante la comparación de competencias en programación de alumnos de primer ingreso y avanzados.
- Proponer una métrica para evaluar alumnos de acuerdo a esfuerzo que les supone una tarea (proceso) y evaluar su fiabilidad.

Para ello, diseñamos un estudio con alumnos de primer ingreso y avanzados de Ingeniería en Computación en el que se les propone resolver un ejercicio de programación mientras se graban sus acciones con el *ResearchLogger* y luego se analizan con el *ResearchAnalyzer*. Como se detalla más adelante, el problema elegido proviene de una batería de ejercicios estandarizados y clasificados previamente por la conocida *Caribbean Online Judge* (COJ) [1].

Independientemente de su grado de avance en la carrera, un estudiante de programación se enfrenta durante la elaboración de código a problemas de origen semántico o sintáctico, lo que ocasiona que realicen búsquedas con relación a sintaxis de lenguajes de programación. Por el tipo de sujetos que participarán, debemos elegir un problema de nivel uno y se

[1] http://coj.uci.cu/index.xhtml

espera que las consultas realizadas sean mayormente de dudas a nivel sintáctico.

Todo esto ha motivado las siguientes hipótesis de trabajo, que intentaremos probar, descartar o modificar de acuerdo a los resultados obtenidos:

H1: Es posible monitorear el proceso de resolución de un ejercicio de programación y el apoyo en web que se necesita para resolver un problema encontrado durante el desarrollo.

H2: Es posible obtener datos que nos permitieran explorar la posibilidad de evaluar el esfuerzo de un estudiante.

H3: El esfuerzo realizado por un estudiante difiere de su resultado en lineas de código.

Al ser este un estudio de caso se realizó un análisis descriptivo y cuantitativo, ya que el tamaño de la muestra no es tan grande los datos deben considerarse solo exploratorios.

IV-A. Indicadores de esfuerzo

Para el caso concreto de sesiones de programación, nos interesa analizar los siguientes indicadores del proceso que consideramos importantes para estudiar las dificultades encontradas y soluciones a las que llegan los alumnos:

- ***B:*** Realizó búsquedas.
- ***C:*** Número de Compilaciones.
- ***BR:*** Número de búsquedas relevantes
- ***PCR:*** Número de páginas consultadas relevantes, entiéndase *relevantes* como búsquedas o páginas consultas relacionadas a el problema o a software.
- ***BI:*** Número de búsquedas irrelevantes.
- ***PCI:*** Número de páginas consultadas irrelevantes, entiéndase *irrelevantes* como búsquedas o páginas consultas NO relacionadas a el problema o a software.
- ***RLA:*** Realizó la lectura del archivo.
- ***RCI:*** Realizó el problema manera parcial, es decir presenta al menos una impresión correcta (ver VI-A).

IV-B. Métrica

Se propone la siguiente *métrica de esfuerzo de programación*:

- Promedio de indicador:

$$\bar{x}(X) = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{1}$$

Donde X corresponde al indicador que se esta analizando, n corresponde al número de estudiantes y x_i corresponde al valor obtenido por el estudiante en el respectivo indicador.

- ***EB***: Evaluación de indicador B

$$EB = \begin{cases} 1 & \text{si } B \text{ se cumple} \\ 0 & \text{si } B \text{ NO se cumple} \end{cases} \tag{2}$$

- ***EC***: Evaluación de indicador C

$$EC = \begin{cases} 1 & \text{si } C \geq \bar{x}(C) \\ 0 & \text{si } C < \bar{x}(C) \end{cases} \tag{3}$$

- ***EBR***: Evaluación de indicador BR

$$EBR = \begin{cases} 1 & \text{si } BR \geq \bar{x}(BR) \\ 0 & \text{si } BR < \bar{x}(BR) \end{cases} \tag{4}$$

- ***EPCR***: Evaluación de indicador PCR

$$EPCR = \begin{cases} 1 & \text{si } PCR \geq \bar{x}(PCR) \\ 0 & \text{si } PCR < \bar{x}(PCR) \end{cases} \tag{5}$$

- ***EBI***: Evaluación de indicador BI

$$EBI = \begin{cases} 1 & \text{si } BI < \bar{x}(BI) \\ 0 & \text{si } BI \geq \bar{x}(BI) \end{cases} \tag{6}$$

- ***EPCI***: Evaluación de indicador PCI

$$EPCI = \begin{cases} 1 & \text{si } PCI < \bar{x}(PCI) \\ 0 & \text{si } PCI \geq \bar{x}(PCI) \end{cases} \tag{7}$$

- ***ERLA***: Evaluación de indicador RLA

$$ERLA = \begin{cases} 1 & \text{si } RLA \text{ se cumple} \\ 0 & \text{si } RLA \text{ NO se cumple} \end{cases} \tag{8}$$

- ***ERCI***: Evaluación de indicador RCI

$$ERCI = \begin{cases} 1 & \text{si } RCI \text{ se cumple} \\ 0 & \text{si } RCI \text{ NO se cumple} \end{cases} \tag{9}$$

Finalmente se propone la fórmula ***FFEC*** *(Formula For Effort Compensation)*:

$$(EB + EC + EBR + EPCR + EBI + EPCI + ERLA + ERCI) \times 12{,}5 \tag{10}$$

Donde 12,5 se obtiene de la división de cien entre ocho, normalización para que la fórmula se ubique entre 0 y 100.

Cuadro I
INTERPRETACIÓN DEL PUNTAJE OBTENIDO CON *FFEC*

Puntaje	**Esfuerzo**
menos de 40	Bajo
40 - 75	Medio
mayor 75	Alto

V. PILOTO DE PRUEBA

Antes de llevar a cabo el estudio con estudiantes de la carrera de Ingeniería en Computación realizamos un piloto de prueba; esto nos permitió evaluar diferentes eventualidades que se presentarían durante la recolección de datos final y en qué medida se puede usar nuestra propuesta para monitorear tareas de programación. Se realizaron dos pilotos de prueba con dos estudiantes avanzados de la carrera Ingeniería en Computación.

Tanto para la prueba piloto como para el estudio final fue necesario escoger un ejercicio de programación que no superara los conocimientos de estudiantes de primer ingreso ya que lo que se desea realizar es un contraste entre estudiantes avanzados y estudiantes de primer ingreso. Se optó por realizar ejercicios provenientes de COJ ya que poseen una

clasificación previa de qué tan complejo resulta un ejercicio y de tipos de ejercicios. Los dos ejercicios tomados para los pilotos son "2858-Parsing Binary Strings"[2] y "2815-Easy String Problem"[3].

V-A. *Condiciones para realizar piloto de prueba*

Para poder realizar una recolección de datos homogénea y asegurar la captura de datos que permitan calcular los indicadores propuestos, se impusieron las siguientes condiciones para propiciar un entorno más controlado:

- Los ejercicios deben ser realizados en el lenguaje de programación Python, esto debido a que la recolección de datos se llevó a cabo con los estudiantes del *Instituto Tecnológico de Costa Rica* y es de nuestro conocimiento que ellos trabajaron durante el periodo inicial de su carrera con este lenguaje.
- Se utilizaría únicamente el IDLE de PYTHON para elaboración de código, ya que sus características permiten minimizar el 'auto-completar' frases o código, permitiendo así aportar a la recolección de datos a nivel de teclado.
- En caso de querer realizar una consulta en web solamente se puede utilizar la herramienta Google-Chrome, ya que a partir de pruebas realizadas previamente al usar esta herramienta es posible filtrar búsquedas y páginas visitadas.
- Antes de iniciar el *ResearchLogger* para captura de datos, se debe haber guardado el archivo con el nombre del estudiante.

Luego de realizar el piloto de prueba numero uno con el problema "2858-Parsing Binary Strings", se obtuvieron los siguientes resultados, plasmados en la Figura 2.

Sujeto	Herramientas	Milisegundos por recurso	%Recurso
USER1	chrome	493986	18.43%
	pythonw	2162654	80.67%
	noprocname	328	0.01%
	WINWORD	23876	0.89%
	total	2680844	100.00%
USER2	chrome	790690	15.83%
	explorer	168128	3.37%
	noprocname	16780	0.34%
	pythonw	4017729	80.43%
	WINWORD	1970	0.04%
	total	4995297	100.00%

Desarrollo del usuario		
	USER1	USER2
Tiempo total de sesión	44.68073333	83.25495

Distribución en minutos de recursos principales		
	USER1	USER2
chrome	8.2331	13.17817
pythonw	36.04423333	66.96215
noprocname	0.005466667	0.279667
WINWORD	0.397933333	0.032833
explorer	0	2.802133

Figura 2. Resultados del primer piloto sobre el problema 2858.

Una vez que se realizó el piloto de prueba numero uno, gracias a la retroalimentación de los estudiantes avanzados y el análisis de datos se llegó a las siguientes conclusiones:

- Fue necesario cambiar la especificación del problema a resolver y el idioma de inglés a español.
- El promedio de tiempo para la resolución del problema es de aproximadamente 50 minutos, ya que aunque la sesión del estudiante *"USER"* tuvo una duración más prolongada, tanto él como el estudiante *"USER1"* finalizaron el ejercicio en los primeros 45 minutos, esto se pudo corroborar a partir del análisis de los datos generados, en donde se observa que el estudiante *"USER2"* dobló el tiempo promedio debido a que quería mejorar su solución preliminar.
- Fue posible realizar un filtrado del número de compilaciones si el estudiante compila con F5.

Durante la realización del piloto se discutió acerca de si eliminar uno de los dos problemas principales que comprende el piloto, el primero es resolver el problema extraído de COJ y el segundo comprende la lectura del archivo de texto para evaluar el ejercicio resuelto. Finalmente se acordó no eliminar la sección de lectura del archivo de texto, esto debido a que seria un buen indicador ver cuanto esfuerzo (Búsquedas en web) requiere un estudiante para resolver dicho problema.

Piloto de Prueba número dos con el problema "2815-Easy String Problem", se obtuvieron los resultados que se muestran en la Figura 3.

Sujeto	Herramientas	Milisegundos por recurso	%Recurso
USER1	chrome	89344	12.06%
	pythonw	645327	87.13%
	WINWORD	5969	0.81%
	total	740640	100.00%
USER2	chrome	711357	24.23%
	explorer	248550	8.47%
	WINWORD	890	0.03%
	pythonw	1861456	63.41%
	py	781	0.03%
	noprocname	112357	3.83%
	total	2935391	100.00%

Desarrollo del usuario		
	USER1	USER2
Tiempo total de sesión	12.344	48.92318

Distribución en minutos de recursos principales		
	USER1	USER2
chrome	1.489066667	11.85595
pythonw	10.75545	31.02427
WINWORD	0.099483333	0.014833
explorer	0	4.1425
py	0	0.013017
noprocname	0	1.872617

Figura 3. Resultados del segundo piloto sobre el problema 2815.

Posterior a la realización del piloto de prueba número dos, gracias a la retroalimentación de los estudiantes avanzados y el análisis de datos se llegó a las siguientes conclusiones:

- Como se previó ambos estudiantes resolvieron el ejercicio antes de los 50 minutos, por lo cual se tomará como tiempo límite para los piloto final.
- La redacción se mejoró y permitió que ambos estudiantes realizaran la lectura de el archivo de texto.
- El conocimiento previo de uno de los estudiantes, le permitió realizar el ejercicio en menor cantidad de tiempo, esto debido a que el estudiante contaba con conocimientos más específicos en recuperación de la información textual.

VI. Ejecución del estudio

Previo al inicio de la sesión de programación, se solicitó a cada uno de los participantes llenara un formulario sobre datos demográficos, preguntas de calificación simple como la edad, sexo y el grado académico del estudiante. Todos los participantes tenían conocimientos de programación en Python, y solo los estudiantes de primer ingreso no habían realizado algún tipo de programación competitiva tipo COJ.

[2]http://coj.uci.cu/24h/problem.xhtml?pid=2858
[3]http://coj.uci.cu/24h/problem.xhtml?pid=2815

VI-A. Ejercicio Usado

Como se explica en la Sección V, se escogió un ejercicio de COJ que fue previamente evaluado y finalmente se dictaminó que el ejercicio a realizar sería el de '2815-Easy String Problem', ejercicio de nivel uno de manejo de 'strings', estructura de datos utilizada para guardar caracteres que pueden ser vistos como texto. A este problema se le realizaron algunas modificaciones en su especificación ya que conocer si el estudiante tiene manejo o no del inglés está fuera de los límites de esta investigación, por lo que se tradujo el problema al español, se eliminaron las restricciones de cantidad de casos de entrada y del largo de las cadenas de caracteres; también se agregó al problema la lectura de un archivo de texto el cual contendría los casos de prueba para ejecutar el problema y a la especificación de entrada y salida se les agregó una línea más de prueba.

VI-B. Condiciones para ejecutar el estudio

- Los ejercicios deben ser realizados en el lenguaje de programación Python.
- Se utilizaría únicamente el entorno de programación IDLE de PYTHON para elaborar el código, ya que sus características permiten minimizar el 'auto-completar' frases o código, lo cual no es capturado tecla por tecla y propicia pérdida de información.
- En caso de querer realizar una consulta web solamente se puede utilizar la herramienta Google-Chrome, ya que a partir de pruebas realizadas previamente es posible filtrar búsquedas y páginas visitadas.
- Se debe guardar el archivo en el Escritorio de Windows.
- Solamente se puede compilar el código generado con la tecla F5, esto para obtener un indicador de número de compilaciones durante la sesión de programación de manera más confiable.

VI-C. Desarrollo de la sesión experimental

Se instaló la nueva versión del *ResearchLogger* en nueve computadoras de uno de los laboratorios de computación del Centro Académico de Alajuela, una de las sedes del Tecnológico de Costa Rica, posteriormente realizamos sesiones de programación con ocho estudiantes de primer ingreso y siete estudiantes avanzados. Estas tuvieron una duración de una hora, debido a que los estudiantes cuentan con otras asignaciones académicas y se tuvo que coordinar con ellos para realizar las sesiones de programación. En primera instancia a los estudiantes se les entregó el formulario de consentimiento sobre el uso de datos (que será para fines académicos, anonimizados y no se divulgarán sus datos personales), a continuación los estudiantes llenaron el formulario demográfico y procedimos con las explicaciones de los supuestos (ver Subsección VI-B); una vez que terminamos la explicación los estudiantes realizaron la sesión de programación. Durante la sesión solo se les permitió hacer uso de hojas en blanco que fueron asignadas previamente por los investigadores para modelado del problema o apuntes de algún tipo y no se permitió el uso de alguna tecnología de bolsillo como celulares para evitar pérdida de datos complementarios a los capturados por teclado y mouse pero que son de gran valor al momento de analizar los datos.

VII. Resultados

Inicialmente se recopilaron datos con dos grupos heterogéneos de estudiantes de la carrera de Computación del Tecnológico de Costa Rica: ocho estudiantes avanzados y siete estudiantes de primer ingreso. Posteriormente debieron descartarse algunos sujetos y solo se utilizaron los datos de seis estudiantes de cada grupo, tres participantes son de sexo femenino y todos entre diecisiete y veintidós años de edad. Sujetos *A01, A02, A03, A04, A12,A14* (Estudiantes que cursan el ultimo año de la carrera de computación) y sujetos *P01, P03, P11, P12, P14, P15* (Estudiantes que cursan el primer año de la carrera de computación).

Se recuperaron datos del proceso de la elaboración de texto y datos e imágenes relacionadas con las interrupciones de mouse, pero estos últimos no están dentro del alcance de esta investigación.

En el caso de los estudiantes avanzados, aunque todos contaban con experiencia en programación competitiva, uno de los estudiantes había participado en olimpiadas de programación, y contaba con conocimiento extracurricular debido a que formó parte de un campamento de programación competitiva realizado en Cuba[4], lo cual explicó que solucionara el problema en un tiempo excepcionalmente corto. Se eliminó uno de las sesiones de programación debido a un fallo en la captura de las interrupciones de teclado, lo que ocasionó una mala construcción del archivo de texto estructurado de la carpeta *detailed_log* (descrito en la Sección III).

En el caso de los estudiantes de primer ingreso se eliminaron dos de las sesiones programación recolectadas, una debido a que el estudiante cerró la ventana del *ResearchLogger*, lo cual produjo que la sesión de programación solo captura 0.14 minutos; en cuanto a la sesión restante se eliminó debido a que el estudiante realizó toda la programación en su hoja de apoyo, por lo cual si quisiéramos evaluar su proceso de programación precisaríamos de una metodología observacional en lugar de la técnica de *keylogging*.

Los datos recopilados fueron procesados por nuestro *script* "logs_interpreter.py" que generó la siguiente información: número de compilaciones de los sujetos de prueba, el tiempo total de sesión de cada uno, número de búsquedas realizadas, número de páginas consultadas, y una distribución de tiempo por cada recurso usado durante la sesión de programación, ver Figuras 2 y 3, también se generaron gráficos de barras para comparar estos datos, ver Figura 4. Todos estos datos se muestran simultáneamente en una hoja de calculo de excel.

Junto con una reconstrucción parcial del texto digitado se realizó un filtrado de las interrupciones de teclado por recurso utilizado y un filtrado de búsquedas digitadas y de páginas visitadas por el estudiante, como se puede ver en las Figuras 5 y 6.

[4]http://www.uci.cu/en/university/news/caribbean-acm-icpc-training-camp-opens

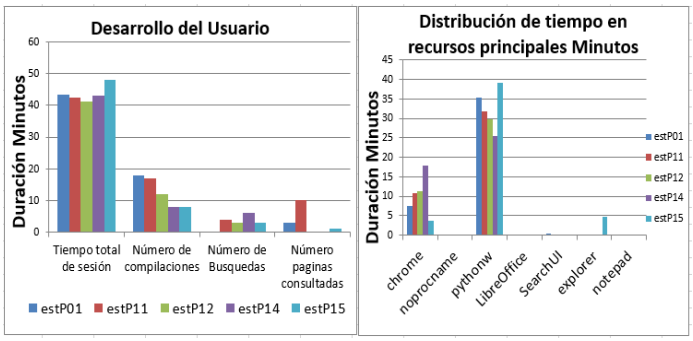

Figura 4. Gráficos generados por nuestra herramienta.

Teclas por ventana			Tiempo en recurso
Recurso y Herramienta:	chrome	Nueva pestaña - Google Chrome	0.140883333
leer "space" un "space" archivo "space" "downarrow" "downarrow" "downarrow" "downarrow" "downarrow" "return"			
l "2e" r un archivo "5downarrow"			

Figura 5. Reconstrucción de texto generado por nuestra herramienta.

Filtrado de busquedas	Paginas consultadas
manejo de strings en python	Trabajar con archivos de texto en Python
manejo de archivos en python	Trabajar con archivos de texto en Python
	String en Python - ChuWiki
	parsing - How do I parse a string to a floa
	Leyendo linea especifica
	Leyendo linea especifica
	String en Python - ChuWiki
	Capítulo 11. Manejo de archivos (Algoritn

Figura 6. Datos generados por nuestra herramienta.

A partir de la reconstrucción de texto se puede observar que el estudiante *P01* dedicó la mayor parte de su sesión de programación a realizar la lectura del archivo de texto, finalmente desistió y procedió a realizar búsquedas con respecto a manejo de *'string'*; su *script* de python solo contenía la siguiente línea de código: `y= map.split()`. En cuanto al estudiante *P14* se observa que durante su sesión de programación trabajó durante toda la sesión en manejo de *'string'*.

Con respecto a la resolución del problema programado de COJ solo un estudiante de primer ingreso fue capaz de resolver el ejercicio completo mientras que de los estudiantes avanzados solo un estudiante no consiguió realizar el ejercicio en su totalidad. Esto muestra una clara diferencia en la capacidad programadora de cada grupo, en línea con lo esperado.

En el caso del lenguaje de programación *Python*, la lectura de un archivo comprende a lo sumo tres líneas de código si agregamos una impresión en pantalla. En el grupo de estudiantes avanzados todos fueron capaces de realizar la lectura del archivo, salvo uno de ellos que generó una solución que proporcionaba un resultado incorrecto y otro realizó una solución incompleta ya que no generaba ningún dato de salida como solicitaba el problema. De los estudiantes de primer año solo uno logró la lectura correcta del archivo, uno fue capaz de abrir el archivo e imprimirlo y los restantes no consiguieron abrirlo ni leerlo.

Una vez realizado el análisis general de los datos, calculamos manualmente la métrica *FFEC* propuesta en la Sección IV-B; realizamos el cálculo en una hoja de cálculo y normalizamos los resultados a una escala de cero a cien.

Calculo de *Promedios*:

Cuadro II
ESTUDIANTES PRIMER INGRESO, PROMEDIOS SEGÚN INDICADOR

	P01	P03	P11	P12	P14	P15	Promedio
C	18	0	17	12	8	8	10,5
BR	4	8	6	3	7	3	5,167
BI	0	0	1	0	0	0	0,167
PCR	3	1	8	0	0	1	2,167
PCI	0	0	2	0	0	0	0,333

Cuadro III
ESTUDIANTES AVANZADOS, PROMEDIOS SEGÚN INDICADOR

	A01	A02	A03	A04	A12	A14	Promedio
C	25	31	24	19	9	21	21,5
BR	6	7	8	18	5	0	7,33
BI	0	0	0	0	0	0	0
PCR	4	8	5	18	5	0	6,67
PCI	0	0	0	0	0	0	0

En cuanto al indicador de Búsquedas, relevantes e irrelevantes, se verificó el texto reconstruido para encontrar búsquedas adicionales que no hubieran sido capturadas automáticamente (ver Sección III) y se añadieron a *BR cuatro, dos y una* búsqueda a los estudiantes *P01, P11, P14*, respectivamente, y *cinco, seis, nueve y dos* búsquedas a los estudiantes *A02, A03, A04, A12*, respectivamente.

Cuadro IV
ESTUDIANTES PRIMER INGRESO, FÓRMULA *FFEC*

	P01	P03	P11	P12	P14	P15
EB	1	1	1	1	1	1
EC	1	0	1	1	0	0
EBR	0	1	1	0	1	0
EPCR	1	1	1	0	0	0
EBI	1	1	0	1	1	1
EPCI	1	1	0	1	1	1
ERCI	0	0	0	1	1	0
ERLA	0	0	0	1	1	0
Esfuerzo 1 a 8	5	5	4	6	6	3
Normalización 0 a 100	62,5	62,5	50	75	75	37,5

Con respecto a los estudiantes avanzados solo un estudiante registró nueve compilaciones, los otros realizaron de entre diecinueve y veinticinco compilaciones durante el proceso de implementación mientras que los estudiantes de primer ingreso registraron menos de diecinueve compilaciones. El rango se encuentra entre ocho y dieciocho compilaciones, esto

Cuadro V
ESTUDIANTES AVANZADOS, FÓRMULA *FFEC*

	A01	A02	A03	A04	A12	A14
EB	1	1	1	1	1	0
EC	1	1	1	0	0	0
EBR	0	1	1	1	0	0
EPCR	0	0	1	1	0	0
EBI	1	1	1	1	1	1
EPCI	1	1	1	1	1	1
ERCI	1	0	1	1	1	1
ERLA	1	1	1	1	1	1
Esfuerzo 1 a 8	6	6	8	7	5	4
Normailizacion 0 a 100	75	75	100	87,5	62,5	50

indica que los avanzados testean más el código durante su producción. Estos resultados se muestran en los Cuadros IV y V.

El uso de los recursos y su distribución por tiempo, mostró que los estudiantes no consumen gran cantidad de tiempo realizando la lectura del programa, el tiempo mayoritario de la sesión se centra en el desarrollo en IDLE y en Google-Chrome realizando búsquedas relacionadas a los problemas presentados. También se puede observar que el tiempo de uso de Google-Chrome es más prolongado en los estudiantes de primer ingreso, lo que indicaría que el conocimiento previo es mayor en los estudiantes avanzados, ver cuadros VI y VII.

El conocimiento previo en los estudiantes avanzados es muy variado debido a que, por su tiempo en la carrera de computación se enfrentan a aprendizajes diversos y pueden o no tener conocimiento relacionado con el problema que se les presentó y con el lenguaje de programación que se asignó esto se muestra en las búsquedas, el estudiante *A14* cuenta con cero búsquedas mientras que el estudiante *A04* cuenta con dieciocho, en el caso de los estudiantes de primer ingreso el rango de las búsquedas varia entre tres y ocho ver cuadros por lo que se puede concluir la cantidad de conocimiento es similar entre estos II y III.

Cuadro VI
DISTRIBUCIÓN EN MINUTOS POR RECURSOS, PRIMER INGRESO.

	P01	P03	P11	P12	P14	P15
chrome	7.577	14.942	10.794	11.295	17.844	3.711
python	35.404	24.085	31.703	29.773	25.364	39.226

Cuadro VII
DISTRIBUCIÓN EN MINUTOS POR RECURSOS, AVANZADOS

	A01	A02	A03	A04	A12	A14
chrome	6.970	9.652	7.277	3.161	9.847	1.376
python	37.004	33.215	35.012	36.705	37.742	10.857

VIII. CONCLUSIÓN

El objetivo general de esta investigación fue estudiar la factibilidad de evaluar a los alumnos por el proceso de desarrollo y no solo por el producto que generan. Para ello se tomaron datos con la herramienta *ResearchLogger* en un ambiente controlado y se realizó un análisis de los datos con indicadores específicos del proceso de programación. De las hipótesis planteadas podemos concluir que: **H1:** *ResearchLogger* es aplicable al monitoreo del *proceso de programación* y fue posible capturar información sobre los indicadores de esfuerzo de los estudiantes.**H2:** Las herramientas *ResearchLogger* y *ResearchAnalyzer* son funcionales para realizar análisis de procesos de programación que pueden ser aplicados en evaluaciones de ámbito académico como pruebas cortas y exámenes que se asemejen a las sesiones de programación y que cuenten con una duración preestablecida. **H3:** Con los datos y resultados obtenidos de las sesiones de programación de los sujetos *P01* y *P14* es posible observar que la métrica *FFEC* compensa el esfuerzo del estudiante, ya que inicialmente ambos obtendrían una nota de cero si solo se evalua su resultado en el *script* de programación realizado; con la métrica FFEC los resultados serían de sesenta y dos punto cinco para el estudiante *P01* y de setenta y cinco para el estudiante *P14*, (ver cuadro IV), lo que los ubica en un esfuerzo medio, (ver cuadro I) , en relación a los demás estudiantes de primer ingreso. Por lo tanto se concluye que el resultado final de un ejercicio programado sí difiere del esfuerzo que realiza un estudiante.

Adicionalmente, la reconstrucción parcial de texto y la otra informacion recolectada brinda la posibilidad de realizar un análisis cualitativo sobre cada sujeto.

Como trabajo futuro se proponen mejoras a estas herramientas, como la automatización de la compresión de los datos utilizando por ejemplo aprendizaje automático.

REFERENCIAS

[1] C. Daly and J. Waldron, "Assessing the assessment of programming ability," in *ACM SIGCSE Bulletin*, vol. 36, no. 1. ACM, 2004, pp. 210–213.

[2] J. Bennedsen and M. Caspersen, "Assessing process and product: a practical lab exam for an introductory programming course," *Innovation in Teaching and Learning in Information and Computer Sciences*, vol. 6, no. 4, pp. 183–202, 2007.

[3] S. Göpferich, A. L. Jakobsen, and I. Mees, *Behind the mind: Methods, models and results in translation process research*. Samfundslitteratur, 2009, vol. 37.

[4] I. Lacruz and R. Jääskeläinen, *Innovation and Expansion in Translation Process Research*. John Benjamins Publishing Company, 2018.

[5] A. L. Jakobsen, "Translation drafting by professional translators and by translation students," *Copenhagen studies in language*, no. 27, pp. 191–204, 2002.

[6] K. Norman, *Cyberpsychology: An introduction to human-computer interaction*. Cambridge university press, 2017.

[7] B. Curtis, E. Soloway, R. Brooks, J. Black, K. Ehrlich, and R. Ramsey, "Software psychology: The need for an interdisciplinary program," *Proceedings of the IEEE*, vol. 74, no. 8, pp. 1092–1106, 1986.

[8] P. Robinson and J. Carroll, "An online system for monitoring and assessing the programming process," 2016.

[9] Y. Yokoyama and H. Egi, "Encouraging system for teaching assistants to advise students during programming exercises."

[10] V.-A. Romero-Zaldivar, A. Pardo, D. Burgos, and C. D. Kloos, "Monitoring student progress using virtual appliances: A case study," *Computers & Education*, vol. 58, no. 4, pp. 1058–1067, 2012.

[11] P. Estrella, R. Lafuente, and A. García., "Prototyping a new keylogging tool for translation process research," in *1st International Congress on Translation, Interpreting and Cognition, Mendoza, Argentina*, 2017.